AMENDMENTS TO THE CLAIMS

This listing of claims replaces all prior versions, and listings, of claims in the application:

1. (Currently Amended) A method for analyzing a program, comprising:

providing an application builder that receives source code instructions and determines the minimum amount of code that is required by a program;

determining a set of functions required by the program by performing local type constraint analysis at intermediate language instruction level to determine which functions have the potential of being executed; and

determining a call path that may reach a function containing such instruction.

2. (Original) The method of Claim 1, further comprising:

analyzing a program instruction that accesses an object field, wherein the analysis is performed locally to an object instantiation.

3. (Original) The method of Claim 1, further comprising: analyzing a program instruction that accesses an array element locally to an array instantiation.

4. (Original) The method of Claim 1, further comprising:

analyzing a program instruction that accesses runtime information for a local runtime symbol usage.

5. (Original) The method of Claim 1, further comprising:

analyzing a program instruction within an exception handler performed locally to an exception instruction.

6. (Original) The method of Claim 1, further comprising:

declaring possible return types of native functions, where a type analysis of intermediate language instruction is not possible.

7. (Original) The method of Claim 6, wherein the set of functions may be in a single program image.

8. (Currently Amended) A computer-readable medium storing computer-executable process steps of a process for analyzing a program, comprising:

determining a set of functions required by the program by performing local type constraint analysis at intermediate language instruction level to determine which functions have the potential of being executed and eliminating unused functions whereby the program is analyzed recursively to determine which functions are called throughout the program; and

determining a call path and a value graph that may reach a function containing such instruction.

9. (Original) The computer readable medium of Claim 8, further comprising:

analyzing a program instruction that accesses an object field, wherein the analysis is performed locally to an object instantiation.

10. (Original) The computer readable medium of Claim 8, further comprising:

analyzing a program instruction that accesses an array element locally to an array instantiation.

11. (Original) The computer readable medium of Claim 8, further comprising:

analyzing a program instruction that accesses runtime information for a local runtime symbol usage.

12. (Original) The computer readable medium of Claim 8, further comprising:

analyzing a program instruction within an exception handler performed locally to an exception instruction.

13. (Original) The computer readable medium of Claim 8, further comprising:

declaring possible return types of native functions, where a type analysis of intermediate language instruction is not possible.

14. (Original) The computer readable medium of Claim 13, wherein the set of functions may be in a single program image.

15. (Currently Amended) A method for analyzing a program, comprising:

determining an object type that may exist at an execution point of the program and evaluating all possible object types that are created at every instruction of a program and carrying the object types through a stack evaluation, wherein this enables determination of a possible virtual function that may be called and allows determination and solutions to program bottlenecks while minimizing virtual functions and dead codes.

16. (Original) The method of Claim 15, further comprising:

creating a call graph at a main entry point of the program; and recording an outgoing function call within a main function.

17. (Original) The method of Claim 16, further comprising:

analyzing possible object types that may occur at any given instruction from any call path for a virtual call.

18. (Original) The method of Claim 17, wherein possible object types are determined by tracking object types as they pass through plural constructs.

19. (Original) The method of Claim 15, further comprising:

calling into function generically for handling specialized native runtime type information.

20. (Currently Amended) A computer-readable medium storing computer-executable process steps of a process for analyzing a program, comprising:

determining an object type that may exist at an execution point of the program enabling determination of possible virtual function that may be called and evaluating all possible object types that are created at every instruction of a program and carrying the object types through a stack evaluation, and providing visibility to functions which are required and also to the chain of dependencies. ~~wherein this enables determination of possible virtual functions that may be called~~.

21. (Original) The computer readable medium of Claim 20, further comprising:

creating a call graph at a main entry point of the program; and recording an outgoing function call within a main function.

22. (Original) The computer readable medium of Claim 21 further comprising:

analyzing possible object types that may occur at any given instruction from a call path for virtual calls.

23. (Original) The computer readable medium of Claim 22, wherein possible object types are determined by tracking object types as they pass through plural constructs.

24. (Original) The computer readable medium of Claim 20, further comprising:

calling into functions generically for handling specialized native runtime type information.

25. (Withdrawn) A method for building an application, comprising:

intermediate language receiving instructions and interpreting and analyzing same;

determining optimum code requirement; and compiling native processor functions comprising native functions that return a declared type; native functions that return a set of types and return functions that vary according to input parameters.

26. (Withdrawn) The method of Claim 25, wherein the optimum code is determined by performing a flow-sensitive analysis that determines possible types of objects that may exist at any instruction of a program.

27. (Withdrawn) The method of Claim 26, wherein based on a set of constraints, virtual functions that have the potential of being executed are determined.

28. (Withdrawn) A computer-readable medium storing computer-executable process steps of a process for building an application, comprising:

intermediate language receiving instructions and interpreting and analyzing same;

determining optimum code requirement; and compiling native processor functions comprising native functions that return a declared type; native functions that return a set of types and return functions that vary according to input parameters.

29. (Withdrawn) The computer readable medium of Claim 28, wherein the optimum code is determined by performing a flow-sensitive analysis that determines possible types of objects that may exist at any instruction of a program.

30. (Withdrawn) The computer readable medium of Claim 29, wherein based on a set of constraints, virtual functions that have the potential of being executed are determined.

31. (Original) The method of Claim 1, wherein the program runs in a managed runtime environment.

32. (Original) The computer readable medium of Claim 8, wherein the program runs in a managed runtime environment.

33. (Original) The method of Claim 15, wherein the program runs in a managed runtime environment.

34. (Original) The computer readable medium of Claim 20, wherein the program runs in a managed runtime environment.

35. (Withdrawn) The method of Claim 25, wherein the program runs in a managed runtime environment.

36. (Withdrawn) The computer readable medium of Claim 28, wherein the program runs in a managed runtime environment.

37. (Withdrawn) A method for determining variable size in a program, comprising: analyzing program function calls recursively and tracking variable size; educing variable size of program function calls for program execution.

38. (Withdrawn) The method of Claim 37, wherein if a variable is discrete, then it is hard coded to a single value.

39. (Withdrawn) The method of Claim 37, wherein if a first variable is assigned to a second variable, then a size constraint of the first variable is merged into a size constraint of the second variable.

40. (Withdrawn) A computer-readable medium storing computer-executable process steps of a process for determining variable size in a program, comprising:

analyzing program function calls recursively and tracking variable size; and reducing variable size of program function calls for program execution.

41. (Withdrawn) The computer readable medium of Claim 40, wherein if a variable is discrete, then it is hard coded to a single value.

42. (Withdrawn) The computer readable medium of Claim 40, wherein if a first variable is assigned to a second variable, then a size constraint of the first variable is merged into a size constraint of the second variable.

43. (Withdrawn) A method for reducing empty function calls in a program, comprising:

determining if a call is made to an empty function; and removing code that creates exceptions where exceptions are not handled; and removing code that checks values where the values can be determined in advance.

44. (Withdrawn) A computer-readable medium storing computer-executable process steps of a process for reducing empty function calls in a program, comprising:

determining if a call is made to an empty function; and removing code that creates exceptions where exceptions are not handled; and removing code that checks values where the values can be determined in advance.

45. (Withdrawn) A method for reducing throw instruction without exception handlers in a program, comprising:

determining if there are any throw instructions without exception handlers; removing throw instructions without exception handlers.

46. (Withdrawn) A computer-readable medium storing computer-executable process steps of a process for reducing throw instruction without exception handlers in a program, comprising:

determining if there are any throw instructions without exception handlers; and removing throw instructions without exception handlers.

47. (Withdrawn) A method for discarding comparison instructions in a program, comprising:

analyzing program instructions and tracking integer values; determining if there are any comparison instructions with discrete values in the program; discarding a comparison instruction and code outside of the determined discrete values and executing the program.

48. (Withdrawn) A computer-readable medium storing computer-executable process steps of a process for discarding comparison instructions in a program, comprising:

analyzing program instructions and tracking integer values; determining if there are any comparison instructions with discrete values in the program; discarding a comparison instruction and code outside of the determined discrete values and executing the program.